UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Medición de diferencia de fase entre señales utilizando FPGAs

Paulo Cesar Casa Robles
Tutor: Antonio Rovira Moreno
Ponente: Eduardo Boemo Scalvinoni

Julio 2018

Medición de diferencia de fase entre señales utilizando FPGAs

AUTOR: Paulo Cesar Casa Robles
TUTOR: Antonio Rovira Moreno

Digital System Laboratory
Dpto. Tecnología Electrónica y de Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2018

Resumen (castellano)

Este Trabajo Fin de Grado consiste en el diseño y desarrollo de un circuito prototipo en una FPGA, para medir la diferencia de fase que se genera entre dos señales. El trabajo se ha desarrollado utilizando una tarjeta de evaluación Digilent Nexys 3, la cual utiliza una FPGA Spartan-6 de Xilinx.

Para diseñar e implementar los circuitos sobre la FPGA se ha utilizado el entorno de desarrollo Xilinx ISE Design Suite, que permite la síntesis y análisis de diseños HDL o la realización de simulaciones. Además, VHDL ha sido el lenguaje seleccionado para la realización del trabajo.

El sistema completo pretende utilizar una FPGA para medir el desfase de dos señales, así como la realización de un módulo de test encargado de generar las señales que serán utilizadas para comprobar el correcto funcionamiento del sistema. Este módulo de test permite seleccionar el desfase deseado, con una precisión de 10ns, para dos señales de 160ns de período.

El medidor de fase consiste en una compuerta XOR y un contador de pulsos. El reloj utilizado por el contador es el reloj interno de la Nexys 3, que tiene una frecuencia de 100MHz, por lo tanto, la precisión del sistema medidor de fase es de 10ns.

Los resultados de las medidas se muestran al usuario mediante la utilización de los displays 7-segmentos que vienen integrados en la propia tarjeta de evaluación. Las pruebas realizadas utilizando el módulo de test concluyen que el sistema funciona de forma correcta, aunque queda pendiente la realización de pruebas más exhaustivas.

Tanto el desarrollo de este sistema como los resultados obtenidos se exponen a lo largo de los capítulos de esta memoria. Se proponen además varias soluciones como trabajo futuro.

Palabras clave (castellano)

Medición, Diferencia, Fase, Señal, Tarjeta, Placa, FPGA, Entorno, VHDL

Abstract (English)

This Final Degree Project consists in the design and development of a prototype circuit in an FPGA, to measure the phase difference that is generated between two signals. The project has been developed using a Digilent Nexys 3 evaluation card, which uses a Spartan-6 FPGA from Xilinx.

In order to design and implement the circuits on the FPGA, the Xilinx ISE Design Suite development environment has been used, which allows the synthesis and analysis of HDL designs or simulations. In addition, VHDL language has been selected to carry out the project.

The complete system tries to use an FPGA to measure the phase movement of two signals, as well as the realization of a test module in charge of generating the signals that will be used to check the correct operation of the system. This test module allows to select the phase shift desired, with a precision of 10ns, for two signals of 160ns of period.

The phase meter consists in an XOR gate and a pulse counter. The clock used by the counter is the internal clock of the Nexys 3, which has a frequency of 100MHz, therefore, the accuracy of the phase meter system is 10ns.

The results of the measurements are shown to the user by using the 7-segment displays that are integrated into the evaluation card itself. The tests carried out using the test module conclude that the system works correctly, although more exhaustive tests are pending.

Both, the development of this system and the results obtained, are exposed in the next chapters of this report. Several solutions are also proposed as future work

Keywords (inglés)

Measurement, Difference, Phase, Signal, Card, Board, FPGA, Environment, VHDL.

Agradecimientos

En primer lugar, me gustaría agradecer tanto a mi tutor Antonio Rovira como a mi ponente Eduardo Boemo, la posibilidad de realizar este proyecto. Agradeciendo así mismo el tiempo y esfuerzo invertidos.

Este TFG marca el fin de un ciclo muy especial. Especial por el apoyo de los que ya estaban, y por personas que he podido conocer como nuevos compañeros y amigos que me llevo, en especial a mi compi de laboratorio con el que tanto tiempo hemos pasado dentro de la Universidad.

Por último, me gustaría agradecer a mi familia, en especial a mis padres, pues han sido el mejor y mayor apoyo que he podido tener. Gracias por vuestra ayuda, paciencia, atención, legado y por siempre confiar en mí. Sé que principalmente sin ellos no hubiese logrado estar donde estoy.

Gracias.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

Los equipos encargados de medir la diferencia de fase entre señales tienen múltiples aplicaciones en la industria y en la investigación. Se trata de equipos complejos de elevado coste. En este proyecto se ha buscado implementar sobre FPGA un sistema modesto de medición de fase.

El proyecto cuenta con un módulo encargado de realizar la medición de fase, y de un módulo encargado de mostrar el resultado a través de un siete-segmentos. Para la comprobación del correcto funcionamiento del circuito se ha diseñado un módulo de test encargado de generar señales internas dentro de la FPGA.

El módulo de test genera dos señales de 160ns de período cuyo desfase puede controlarse utilizando los switches que vienen incorporados con la Nexys 3. Este módulo genera las señales dentro de la FPGA y las transmite al medidor de fase sin sacar las señales fuera de la FPGA en ningún momento. Entre las futuras mejoras del sistema se propone la utilización de señales externas para probar el funcionamiento del sistema, que hasta ahora ha sido probado directamente con el módulo test, y con simulaciones.

1.2 Objetivos

A continuación, se explican los objetivos de este trabajo:

- Realizar el diseño, desarrollo e implementación de una aplicación de un prototipo, de un sistema medidor de fase mediante FPGAs, utilizando la tarjeta de evaluación Nexys 3 de Digilent.
- Aprendizaje en diseño con FPGAs y formación en el uso de lenguajes de descripción hardware (VHDL).

- Estudio bibliográfico de los medidores de fase, realización de pruebas sobre la FPGA, así como estudio de posibles mejoras para futuras versiones del medidor de fase.

1.3 Organización

La memoria consta de los siguientes capítulos:

- **Introducción:** Se explican los motivos y los objetivos de este proyecto.

- **Estado del Arte:** Se analiza el estado de la tecnología actual de los medidores de fase y se presentan algunos ejemplos de equipos que actualmente se encuentran en el mercado.

- **Diseño:** Se realiza la descripción del proyecto, explicando las principales decisiones de diseño que han sido tomadas y de qué manera fueron implementadas.

- **Desarrollo:** Se enuncian las cuestiones referentes al progreso de cada parte del trabajo. Se detalla el funcionamiento del código realizado, así como se explican las soluciones aplicadas a los problemas que han ido apareciendo a lo largo de todo el trabajo.

- **Integración, Pruebas y Resultados:** Se muestran las pruebas realizadas para la comprobación del correcto funcionamiento del diseño, analizando los resultados obtenidos, incluyendo las fotografías que muestran el resultado de dichas pruebas.

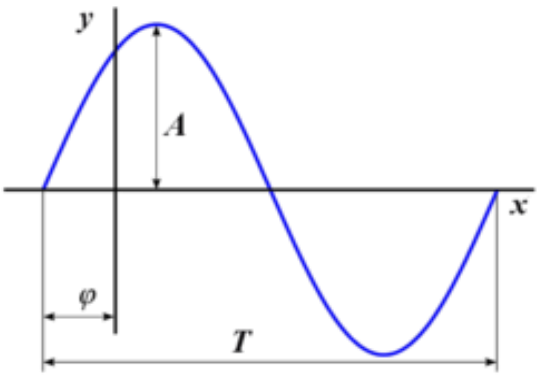
- **Conclusiones y Trabajo Futuro:** Se analizan los resultados obtenidos, se muestran las conclusiones extraídas y se describen varias propuestas que podrían ser implementadas en futuras versiones de este sistema.

2 Estado del arte

La tecnología ha evolucionado rápidamente en los últimos años, y actualmente hay distintos medidores de fase que pueden encontrarse en el mercado. Hay algunos principios básicos que son comunes a todos ellos:

2.1 Señal Sinusoidal

En matemáticas se denomina sinusoide o senoide a la curva que representa gráficamente la función seno y que describe una oscilación repetitiva y suave.



2-1 Señal Sinusoidal

Su forma más básica en función del tiempo (t) es:

$$y(t) = Asen(\omega t + \varphi)$$

2.1.1 Características

- A se corresponde a la amplitud de oscilación
- ω se corresponde a la velocidad angular $\omega = 2\pi f$
- T es el periodo de oscilación $T = 1/f$
- φ es la fase inicial

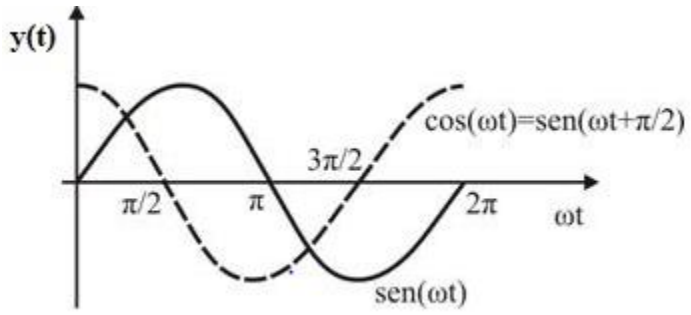
2.1.2 Fase

La fase da una idea del desplazamiento horizontal de la sinusoide. Dos sinusoides están en fase cuando comparten la misma fase, mientras que están en desfase en caso contrario, resultado que una de las sinusoides estará adelantada o atrasada con respecto de la otra.

Para este trabajo se tienen en cuenta únicamente señales sinusoidales que comparten la misma frecuencia.

2.1.3 Diferencia de fase entre señales

Un ejemplo muy claro de diferencia de fase se da entre las señales seno y coseno ya que se encuentran desfasadas $\pi/2$ la una respecto de la otra, tal y como muestra la siguiente figura:



2-2 Señales con distinta fase

2.2 Digital Phase Detector

Se analizaron varias técnicas para la realización de un detector de fase digital, algunas de ellas como el circuito de muestreo y retención, el cual tiene una gran precisión cuando sólo se comparan pequeñas diferencias de fase entre las dos señales. Sin embargo, este tipo de diseños es complicado de implementar en una FPGA y se ha optado por una solución más sencilla.

El método elegido consiste en la utilización de una compuerta lógica OR exclusiva (XOR) y un contador de pulsos. Este detector de fase es adecuado para trabajar con señales de onda cuadrada. Su funcionamiento consiste en que cuando las dos señales

que se comparan están completamente en fase, la salida de la puerta XOR tendrá un nivel constante de cero, mientras que cuando las señales estén desfasadas la salida de la XOR se mantendrá a 1 durante un tiempo equivalente al del desfase entre ambas señales.

2.3 Soluciones del mercado

En el mercado se presenta un rango de soluciones diferentes, comenzando por detectores de fase (e incluso de tensión) de fácil uso y aplicación básica, y terminando con sistemas más sofisticados y de mayor coste económico.

2.3.1 Detectores de fase simples

Como ejemplo encontramos el *MS920-CATU-DETEX*, que se trata de detectores de fase y adicionalmente de baja tensión, contando con una precisión de ±5%.

2-3 MS920-CATU-DETEX

2.3.2 Detectores de fase sofisticados

Estos detectores de fase se basan por tener una mayor precisión que los anteriores y por realizar más funciones a parte de medición del desfase. Por ejemplo, el hardware DSP Logger MX300, utiliza dos sensores de medición simultánea.

2-4 Hardware DSP Logger MX300

2.4 Aplicaciones

Los sistemas de medición de fase son muy útiles en la industria y la investigación, algunas de sus aplicaciones son las siguientes:

- En las máquinas con acoplamiento para el análisis de problemas mecánicos y hacer una distinción entre desbalanceo y desalineación.

- Para medidas de ruido de fase de una señal periódica (variaciones aleatorias de su fase instantánea con respecto a la de una señal ideal) en un cierto rango de frecuencias. Por lo que la idea para medir dicho desfase es la misma. Sin embargo, para esta aplicación se suele aplicar la medición en el dominio de la frecuencia, añadiendo también demoduladores FM y AM.

3 Diseño

Para la realización del módulo de medición de fase se ha utilizado una tarjeta de evaluación Nexys 3, que contiene una FPGA Spartan VI de Xilinx.

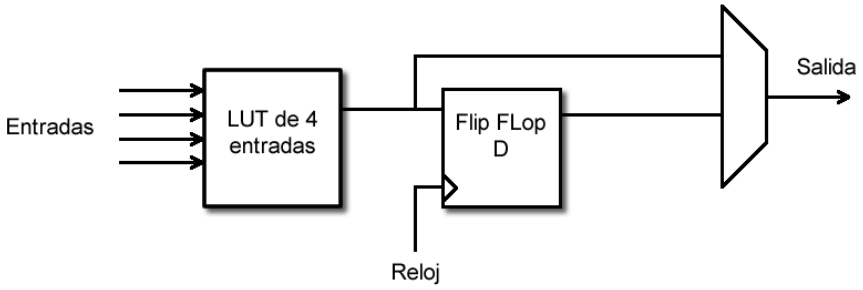
La realización de este proyecto ha requerido comprender brevemente la arquitectura de las FPGAs así como el manejo del lenguaje de descripción hardware VHDL, así como de la herramienta de desarrollo ISE Design Suite.

3.1 Hardware FPGA

Las FPGA, Matrices de Puertas Reprogramables (Field Programmable Gate Array) son dispositivos semiconductores que contienen bloques de lógica, cuya interconexión y funcionalidad se pueden programar, lo que constituye su principal ventaja, ya que al ser reconfigurables (y a pesar de su alto consumo de potencia) son ideales para el desarrollo de prototipos.

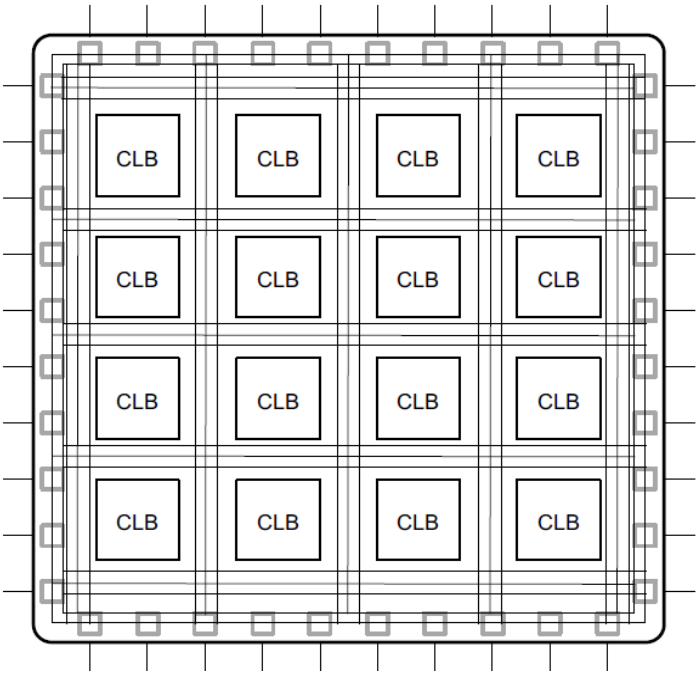
La arquitectura de una FPGA está basada en un gran número de pequeños bloques lógicos programables (CLBs) creados para realizar operaciones lógicas sencillas. Estos bloques a su vez cuentan con biestables que proporcionan funcionalidades síncronas. La enorme flexibilidad de las FPGAs reside en la libertad que se posee a la hora de interconectar dichos bloques.

En la siguiente figura se observa que, los bloques lógicos están compuestos por una lookup table de 4 entradas que puede representar cualquier función lógica combinacional de 4 entradas, y un flip-flop. Además, se observa que tiene una salida que se obtiene o bien de la tabla de búsqueda una vez registrada, es decir que pasa por el flip-flop, o de la tabla de búsqueda sin registrar.



3-1 Diseño de Lógica Configurable

A continuación, se mostrará una de las arquitecturas de una FPGA, que sirve como ejemplo de entre la gran variedad de FPGAs proporcionadas por varias compañías como Xilinx, Altera, Atmel y Lattice. Puede observarse que la FPGA se compone de los bloques configurables de entrada-salida, el conexionado y los bloques lógicos configurables, como el mostrado en la *figura 3-1*. La arquitectura de la FPGA que mostramos a continuación se denominan arquitectura cuadriculada (*figura 3-2*).



3-2 Arquitectura General de una FPGA

En la estructura de la FPGA destacan los siguientes elementos:

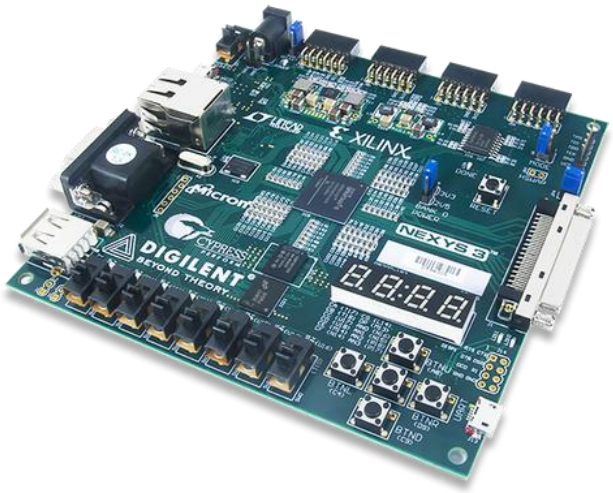
- Bloques lógicos o Look-up Tables (LUTs): bloques que realizan operaciones lógicas.
- Flip-flops (FFs): elementos encargados en registrar el resultado de las operaciones realizadas por las LUTs.
- Hilos de interconexiones programables que conectan los distintos elementos.
- Pads de entrada, salida o ambos (I/O) que son los puertos que intercambian información con el exterior.
- Bloques DSP o unidades aritmético-lógicas embebidas.
- Bloques de memoria.

3.1.1 Spartan-6

La implementación del diseño de este trabajo se ha realizado sobre la tarjeta de evaluación Nexys 3 que contiene una FPGA Spartan-6 de Xilinx.

La Nexys 3 contiene 16MB de memoria RAM y varios puertos de E/S. Puede utilizarse para construir sistemas digitales completos incluidos controladores y procesadores integrados. Cuenta con USB2 de alta velocidad Adept incorporado proporcionando alimentación, programación de FPGA, transferencias de datos de usuario y un reloj de 100 MHz.

En la siguiente figura se puede observarse una imagen de la placa Spartan 6 Nexys 3 utilizada para llevar a cabo las pruebas.



3-3 Placa FPGA Spartan-6 utilizada

3.2 ISE Design Suite

La herramienta utilizada para la implementación del sistema ha sido ISE Design Suite de Xilinx.

Dicha herramienta permite realizar un diseño completo basado en lógica programable, por lo que ésta herramienta incluye el software necesario para realizar todas las etapas que se llevan a cabo en el diseño de un prototipo de un circuito digital.

Estas etapas son las siguientes:

- La entrada de diseño, bien a través de lenguajes de descripción hardware (ABEL, VHDL o Verilog) o bien a través de una captura esquemática.

- Herramienta de verificación para la obtención de una simulación del circuito digital realizado, a niveles funcionales como de estimación de retardos.

- Herramientas de programación, en el cual es posible probar y depurar el sistema sobre el hardware de forma rápida y flexible permitiendo todos los cambios que se necesiten.

- Herramientas de implementación que permite especificar las restricciones o indicaciones que se deban en un circuito para la implementación óptima sobre el dispositivo lógico programable en el que se esté realizando el diseño.

3-4 Logo ISE Design Suite

3.3 Los lenguajes HDL

Un lenguaje de descripción de hardware es un lenguaje de programación destinado a la descripción formal de circuitos electrónicos y de lógica digital. El lenguaje permite modelar la arquitectura y comportamiento de sistemas electrónicos discretos.

Estos lenguajes permiten describir las operaciones del circuito, su diseño, organización y verificación del funcionamiento correcto mediante la simulación. Como los lenguajes de programación concurrente, la sintaxis y la semántica de los lenguajes de descripción de hardware incluyen de forma explícita notaciones para expresar concurrencia.

Sin embargo, en contraste con la mayoría de lenguajes de programación, los lenguajes HDL también incluyen notaciones explícitas de tiempo, lo cual resulta una característica necesaria en el diseño de hardware.

Más adelante veremos que estas notaciones explícitas de tiempo nos serán muy útiles para poder simular el diseño (generar Test Benchs), pero habrá dificultades a la hora de sintetizar el código.

3.3.1 Lenguajes de diseño hardware y FPGAs

Los lenguajes HDLs y dispositivos FPGA permiten a los diseñadores desarrollar y simular rápidamente un circuito digital sofisticado, realizar prototipos y verificar su funcionamiento.

En los últimos tiempos se han convertido prácticamente en imprescindibles y permiten sistemas bastante sofisticados utilizando únicamente un ordenador y pequeña placa FPGA.

3.3.2 VHDL

El lenguaje VHDL es un lenguaje de alto nivel que se utiliza para la descripción del hardware de circuitos digitales, utilizado especialmente con los PLDs, FPGAs, ASICs y similares.

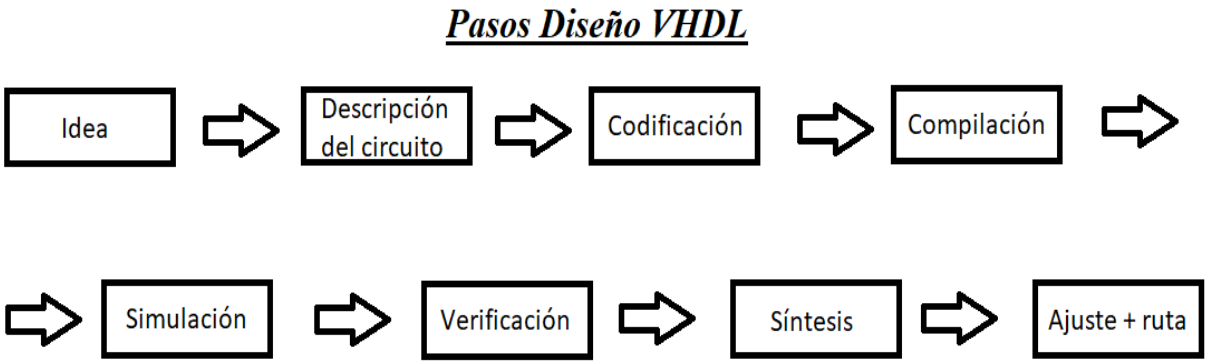
El VHDL permite modelar circuitos digitales a través de tres niveles de descripción: Comportamiento (Funcional), Flujo de Datos y Estructural. Estos tres niveles se describen a continuación:

- **Comportamiento:** Este nivel permite describir el comportamiento del circuito digital sin necesidad de conocer su estructura interna. Este tipo de descripción ha sido utilizado durante este trabajo.

- **Estructural:** Describe los componentes que forman el circuito digital y sus conexiones entre los diferentes componentes, por lo que es necesario conocer la estructura interna del circuito digital.

- **Flujo de datos:** Este nivel describe el comportamiento que deben de tener las diferentes señales del circuito digital.

3.4 Flujo de Diseño



3-5 Pasos en un flujo de diseño VHDL

Se muestran los pasos seguidos en el diseño del trabajo que serán tratados con mayor profundidad en el capítulo de desarrollo.

El objetivo es implementar un circuito que sea capaz de medir la diferencia de fase entre dos señales utilizando una placa FPGA, que también es utilizada para implementar un módulo de test con el que probar el funcionamiento del sistema.

En primer lugar, se ha realizado un esquema sobre el papel de los diferentes bloques que tiene que tener el sistema, para a continuación, diseñar cada uno de esos bloques utilizando lenguaje VHDL.

El programa ISE Design Suite ha sido utilizado para realizar tanto las implementaciones como las simulaciones de dicho código. Las simulaciones han sido realizadas utilizando ISim Simulator.

Tras la síntesis de la descripción VHDL en un conjunto de primitivas o componentes que puedan ensamblarse en la tecnología destino, se llega finalmente a la implementación del proyecto sobre la FPGA.

3.5 Librerías

A continuación, se detalla de forma bastante breve las bibliotecas de la librería IEEE a utilizar. En este caso, se utilizan en los módulos estos tres paquetes:

- El paquete **std_logic_1164** proporciona tipos de señal mejorados.
- El paquete **std_logic_arith** proporciona cómputo numérico.
- El paquete **std_logic_unsigned** proporciona cálculo numérico sin signo.

4 Desarrollo

A continuación, se comenta el desarrollo del trabajo, explicando con detalle la idea de los distintos módulos en los que se ha dividido este trabajo. Se explica la implementación de cada uno de estos módulos y se comentan varios de los problemas que surgieron a lo largo de todo el trabajo.

4.1 Pasos previos

En primer lugar, fue preciso adquirir conocimientos suficientes en VHDL, y familiarizarse con la herramienta ISE Design Suite. En este primer aprendizaje se llevaron a cabo las siguientes acciones:

- Estudio y lectura de libros sobre implementación de funciones y diseños digitales en FPGA. Como por ejemplo *"Diseño Digital: Principios y Prácticas"* ó *"Guide to FPGA Implementation of Arithmetic Functions"*.

- Realización de tutoriales y código, para la familiarización con el entorno de programación. Primero realizando simulaciones y después trabajando directamente sobre la FPGA.

Una vez adquirido este conocimiento, se realizó el diseño del circuito dividiéndolo en tres módulos claramente diferenciados conectados entre ellos utilizando la declaración de componentes dentro del diseño:

```
COMPONENT nombre [IS]
  [GENERIC(lista_parametros);]
  [PORT(lista_de_puertos);]
END COMPONENT nombre;
```

4-1 Declaración Componente en VHDL

4.2 Top Design

El diseño consta de un módulo global, encargado de integrar los diferentes módulos del sistema. Para ello, dentro del *TopDesign* se instanciaron los diferentes módulos antes mencionados, y se "mapearon" las diferentes señales para por último conectar las salidas de los 7-segmentos de la placa FPGA.

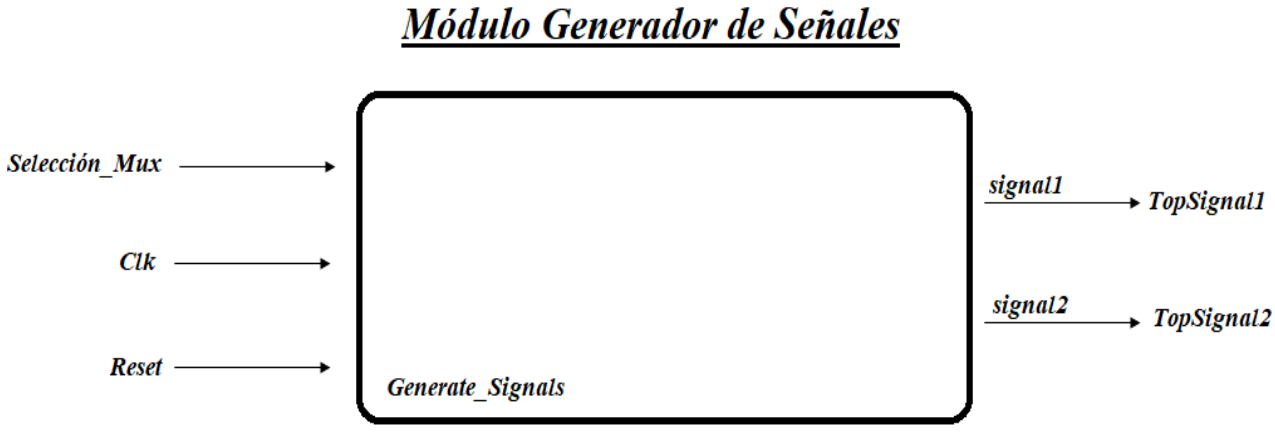
Este breve código, mostrado en la *figura 4.2*, permite que en el *display A* se muestre la selección de los *switches* realizada manualmente, y que en el *display C se* muestre el desfase calculado. Finalmente, el *switch7* se utiliza como un reset global, por lo que cuando dicho *switch* se active, en la pantalla de la FPGA mostrará cero en todos sus displays.

```
Ai(0)<=Switch0;
Ai(1)<=Switch1;
Ai(2)<=Switch2;
Ai(3)<=Switch3;
Bi<="0000";
Ci<= CCi;
Di<="0000";
topsignal1<=signal_aux_1;
topsignal2<=signal_aux_2;
apagado<=Switch7;
```

4-2 Parte del código Top Design

4.3 Módulo Generador de Señales

El módulo generador de señales tendrá el siguiente aspecto esquemático:



4-3 Esquema Módulo Generador de Señales

En este módulo se busca generar dos señales con el mismo periodo, pero con un desfase controlado entre ambas.

Para ello, debemos tener claro el esquema anterior, en donde las salidas serán las dos señales generadas por el módulo.

Dentro del módulo para poder obtener lo que buscamos tenemos que implementar un divisor de frecuencia, un cierto número de FF's y un multiplexor que será el encargado de controlar el desfase.

4.3.1 Divisor de Frecuencia

Se llama divisor de frecuencia a un dispositivo electrónico que divide la frecuencia de entrada en una relación casi siempre entera o racional. La forma de la señal de salida puede ser simétrica o asimétrica. La señal de entrada frecuentemente tiene forma de una onda cuadrada pero también puede ser sinusoidal o de otras formas. Además, suelen estar formados por contadores digitales.

Es interesante para utilizarlo en cualquier aplicación con el tiempo o el periodo por lo que el divisor de frecuencia nos servirá para generar nuestra primera señal, la cual queremos que tenga un periodo específico.

4.3.2 Calculo del periodo

El periodo está relaciona de forma inversa con la frecuencia, y además en el tema de las FGAs, cada una de ellas tiene su frecuencia determinada de trabajo de la tarjeta. En este caso sabemos que el periodo de trabajo de la placa Nexys 3 Spartan-6 es de 10ns o lo que es lo mismo 100MHz.

Una vez conocido este dato, podremos realizar nuestro cometido. Lo podremos realizar utilizando los datos del periodo a partir de la siguiente fórmula:

$$X = \frac{Periodo\ deseado}{Periodo\ reloj\ FPGA}$$

El procedimiento que se ha hecho en este caso es que se ha buscado que el periodo de la señal sea 16 veces el periodo que tiene el reloj interno de la FPGA. Por lo tanto, para el cálculo hay que basarse en la fórmula anterior y añadir posteriormente al resultado una división por 2, como se muestra a continuación:

$$X = \frac{160ns}{10ns} = 16 => X = \frac{16}{2} = 8$$

Como se mencionó antes, el hecho de tener que dividir por dos es necesario ya que justo tendría que cambiar en la mitad la señal.

En este caso, resulta bastante obvio, pero en otros casos en el que se busque un periodo que no sea proporcional al periodo del reloj es de vital importancia realizar el cálculo para no cometer errores.

Una vez tenemos este dato, se pasa a binario y así este valor se corresponderá con el valor que tendrá mi constante a la que se ha llamado *predeterminado1*.

Finalmente, una vez llegue el contador al valor predeterminado, se necesita una compuerta NOT para poner en marcha el divisor de frecuencia, es decir, cambiar de periodo la señal a la que se desea en ese momento. Una vez hecho esto, se vuelve a poner el valor del contador a cero.

En código esta parte se corresponde a:

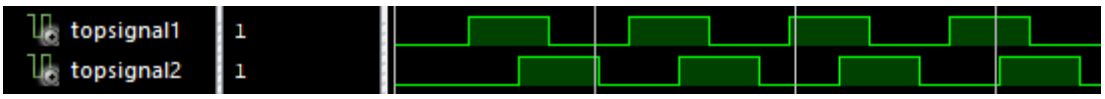
```
signal1<= auxiliar1;

Señal1: process(clk,reset,count)
begin
   if(rising_edge(clk)) then
      if(reset='1') then
         auxiliar1 <='0';
         count <= (others=>'0');
      elsif (count=predeterminado1) then
         auxiliar1 <= not auxiliar1;
         count <= (others=>'0');
      else
         count <= count+'1';
      end if;
   end if;
end process;
```

4-4 Código contador para generar primera señal

4.3.3 Forma de realizar la diferencia de fase

Para generar la segunda señal con un desfase determinado con respecto a la ya creada primera señal (como la vista en la *figura 4-5*) se ha utilizado un método que se analizará a continuación.



4-5 Ejemplo de desfase obtenido entre señales

Este método se basa en generar tantos números de Flip Flops como número de diferencia de fases se quiera que haya entre las dos señales. Utilizaremos este método porque cada vez que se produce un proceso de FF, la salida del FF tardará un periodo de reloj en igualarse a la entrada del FF mencionado.

Por último, para escoger la diferencia de fase, se hará a partir de un multiplexor, el cual, me permitirá escoger (dependiendo de la diferencia que se quiera tener) entre todas las salidas de los FFs mencionados.

Por ejemplo, si queremos que haya una diferencia de fase de hasta 7 periodos a la hora de implementarlo será de la siguiente forma:

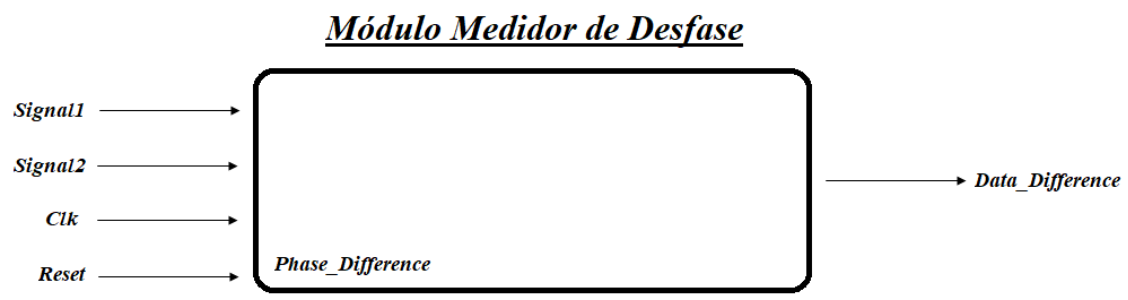
```
mux: Process(seleccion_mux,auxiliar1,salidaFF1,salidaFF2,salidaFF3,salidaFF4,salidaFF5,salidaFF6,salidaFF7)
   begin
      case seleccion_mux is
         when "000" => mux_out<=auxiliar1;
         when "001" => mux_out<=salidaFF1;
         when "010" => mux_out<=salidaFF2;
         when "011" => mux_out<=salidaFF3;
         when "100" => mux_out<=salidaFF4;
         when "101" => mux_out<=salidaFF5;
         when "110" => mux_out<=salidaFF6;
         when "111" => mux_out<=salidaFF7;
         when others => mux_out<='0';
      end case;
end process;

signal2 <= mux_out;
```

4-6 Código Multiplexor que controla la generación de la segunda señal

4.4 Módulo Medidor de De Diferencia de Fase

El módulo medidor se ha diseñado de la siguiente manera:



4-7 Esquema Módulo Medidor de Diferencia de Fase

Este es el módulo encargado de medir el desfase entre señales. Consiste en una puerta XOR cuya salida se mantiene a uno una duración exactamente igual al desfase entre las señales. Y un contador encargado de contar el número de periodos de desfase que haya entre ambas señales.

```
data_difference<=count_xor;
contador_xor:process(clk,entrada_xor,sigDel)
   begin

      if falling_edge(clk) then
            if entrada_xor='1' then
               aux_count<= aux_count+1;
            elsif ( entrada_xor='0' and sigDel='1') then
               count_xor<=aux_count;
               aux_count<=(others=>'0');
            end if;
      sigDel<=entrada_xor;
      end if;


end process;
```

4-8 Código Obtención de la diferencia de fase

Este módulo ha sido el más complicado de realizar y el que más problemas ha planteado y para realizarlo se ha seguido los siguientes pasos:

- Se ha generado un contador el cual cuente el número de periodos de reloj en el momento en el que la señal XOR sea igual a 1 ya que a partir de ese valor puede calcularse la diferencia de fase.

- Se ha diseñado un circuito que actualiza el valor de diferencia de fase al de contador prescindiendo del concepto de *event* ya que dicho concepto sólo es posible utilizar en la generación de un banco de ensayos (*test bench*) o en las señales de reloj.

La solución utilizada se muestra en la *figura 4-8*. Esto se traduce en la creación de una nueva señal que retrase la salida de la XOR y permite calcular su flanco de subida y bajada.

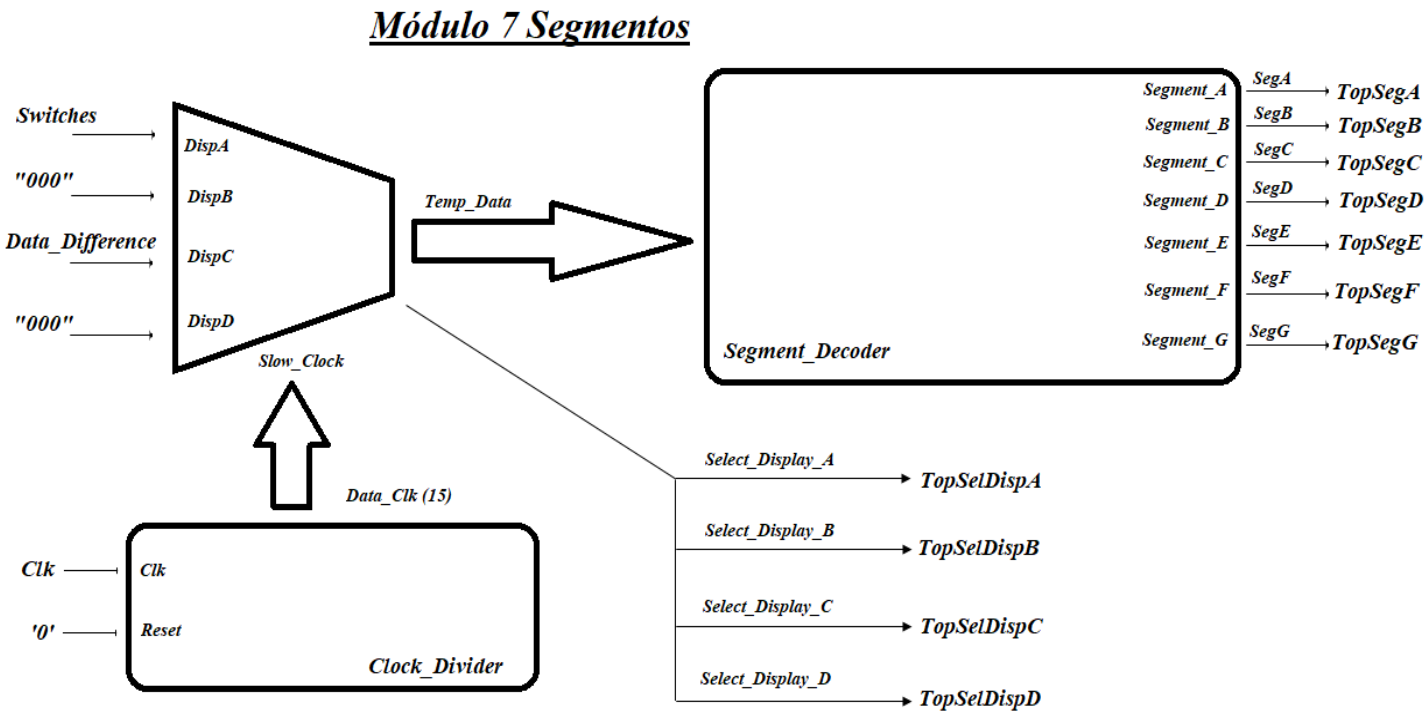
```
elsif ( entrada_xor='0' and sigDel='1') then
   count_xor<=aux_count;
```

4-9 Condición necesaria para actualizar

- Se actualizará medio ciclo de reloj después (5ns) de que la señal XOR pasé de valer '1' a '0', y dicho valor se mantendrá constante hasta que cambie el desfase.

4.5 Módulo 7 Segmentos

Se muestra el esquema que se utilizado para crear el módulo:

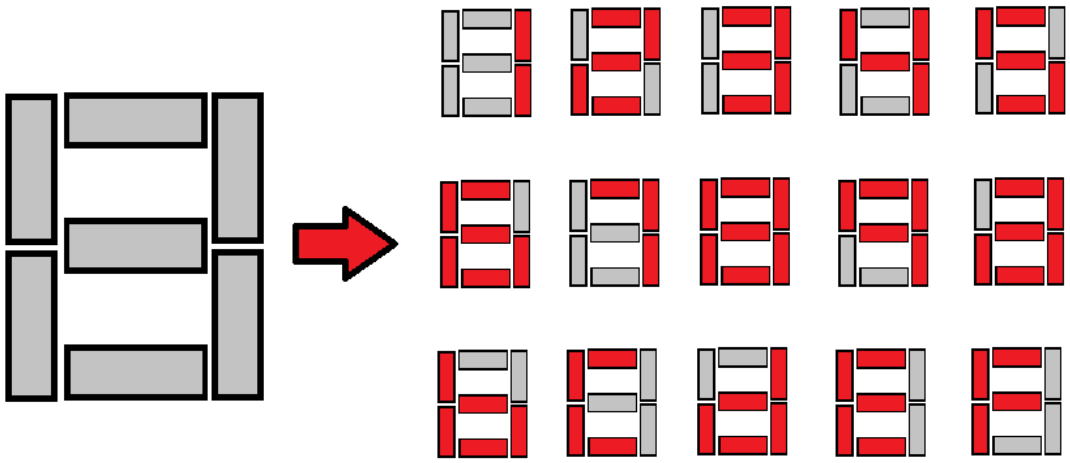


4-10 Esquema Módulo 7 Segmentos

Este módulo busca controlar los 7-segmentos incluidos en la tarjeta de evaluación Nexys 3 para visualizar la diferencia de fase seleccionada para el módulo de test y también la misma diferencia de fase medida por el medidor de fase.

Para generar este módulo 7 segmentos, se han necesitado otros dos submódulos dentro de este (*Segment Decoder* y *Clock Divider*).

Se comenzó creando un decodificador de segmento que decodifica un número binario en siete segmentos, tal que se crea un display de 7 segmentos para números hexadecimales, como el mostrado en la siguiente figura:



4-11 Display de 7 segmentos para números hexadecimales

Es preciso conocer la tabla de verdad para un decodificador de números hexadecimales a 7 segmentos en la FPGA Nexys3, para saber que segmento tiene que estar encendido y cual no. Dicha tabla se muestra a continuación:

Hex Number	Common-Anode	CA	CB	CC	CD	CE	CF	CG	Cathode[6:0]
0	**HIGH**	LOW	LOW	LOW	LOW	LOW	LOW	HIGH	0000001
1	**HIGH**	HIGH	LOW	LOW	HIGH	HIGH	HIGH	HIGH	1001111
2	**HIGH**	LOW	LOW	HIGH	LOW	LOW	HIGH	LOW	0010010
3	**HIGH**	LOW	LOW	LOW	LOW	HIGH	HIGH	LOW	0000110
4	**HIGH**	HIGH	LOW	LOW	HIGH	HIGH	LOW	LOW	1001100
5	**HIGH**	LOW	HIGH	LOW	LOW	HIGH	LOW	LOW	0100100
6	**HIGH**	LOW	HIGH	LOW	LOW	LOW	LOW	LOW	0100000
7	**HIGH**	LOW	LOW	LOW	HIGH	HIGH	HIGH	HIGH	0001111
8	**HIGH**	LOW	LOW	LOW	LOW	LOW	LOW	LOW	0000000
9	**HIGH**	LOW	LOW	LOW	LOW	HIGH	LOW	LOW	0000100
A	**HIGH**	LOW	LOW	LOW	LOW	LOW	HIGH	LOW	0000010
B	**HIGH**	HIGH	HIGH	LOW	LOW	LOW	LOW	LOW	1100000
C	**HIGH**	LOW	HIGH	HIGH	LOW	LOW	LOW	HIGH	0110001
D	**HIGH**	HIGH	LOW	LOW	LOW	LOW	HIGH	LOW	1000010
E	**HIGH**	LOW	HIGH	HIGH	LOW	LOW	LOW	LOW	0110000
F	**HIGH**	LOW	HIGH	HIGH	HIGH	LOW	LOW	LOW	0111000

4-12 Tabla verdad hexadecimal a 7 segmentos

4.5.1 Submódulo Decodificador de Segmento

A la hora de implementar el código se utiliza un multiplexor para elegir un dato de entrada y enviarlo al decodificador de segmentos. Sin embargo, en la tabla anterior se tienen que transformar los números en binario. Los segmentos tienen que estar activos bajos para encenderlos. Este código mencionado se muestra a continuación:

```
begin
process(Digit)
   variable Decode_Data: std_logic_vector(6 downto 0);

   begin
   case Digit is
      when "0000" => Decode_Data:= "1111110"; --0
      when "0001" => Decode_Data:= "0110000"; --1
      when "0010" => Decode_Data:= "1101101"; --2
      when "0011" => Decode_Data:= "1111001"; --3
      when "0100" => Decode_Data:= "0110011"; --4
      when "0101" => Decode_Data:= "1011011"; --5
      when "0110" => Decode_Data:= "1011111"; --6
      when "0111" => Decode_Data:= "1110000"; --7
      when "1000" => Decode_Data:= "1111111"; --8
      when "1001" => Decode_Data:= "1111011"; --9
      when "1010" => Decode_Data:= "1110111"; --A
      when "1011" => Decode_Data:= "0011111"; --B
      when "1100" => Decode_Data:= "1001110"; --C
      when "1101" => Decode_Data:= "0111101"; --D
      when "1110" => Decode_Data:= "1001111"; --E
      when "1111" => Decode_Data:= "1000111"; --F
      when others => Decode_Data:= "0111110"; --error
      end case;

      segment_A <= not Decode_Data(6);
      segment_B <= not Decode_Data(5);
      segment_C <= not Decode_Data(4);
      segment_D <= not Decode_Data(3);
      segment_E <= not Decode_Data(2);
      segment_F <= not Decode_Data(1);
      segment_G <= not Decode_Data(0);
   end process;

end Behavioral;
```

4-13 Código Decodificación de Segmentos

4.5.2 Submódulo Divisor de Reloj

El siguiente submódulo divisor de reloj se encarga de ralentizar el reloj, para que así los datos puedan ser divisibles en la visualización del segmento porque el reloj FPGA al ser de 100MHz es demasiado rápido.

Esto se puede lograr haciendo un contador, tal y como se muestra en la siguiente figura:

```
begin

process(clk,reset)
   variable count: std_logic_vector(15 downto 0):=(others=>'0');
   begin

   if reset='1' then
      count:=(others=>'0');
   elsif clk'event and clk='1' then
      count:= count+1;
   end if;

   data_clk<=count;

end process;

end Behavioral;
```

4-14 Contador Clock Divider

Para ralentizar el reloj en este módulo, se utiliza el bit más significativo de la señal de salida del módulo que coincide con el valor del contador, es decir, el bit que se encuentra en la posición decimosexta de este nuevo valor.

4.5.3 Unión Componentes para obtener el Módulo

En el módulo principal se instancian en componentes los submódulos previamente mencionados para poder utilizarlos dentro de la arquitectura.

En este módulo tendremos una señal que almacenará el bit más significativo del contador anterior y gracias a dicha señal se ralentizará el proceso ya que se encuentra en su lista de sensibilidad.

Seguidamente creamos una variable que se encargará de seleccionar a que display queremos que vaya el valor correspondiente o a los switches o a la medición de

desfase, de lo que más adelante se hablará para saber cómo se obtienen dichos valores que quieren ser mostrados en los diferentes displays de la FPGA. A continuación, se muestra el código previamente explicado:

```
slow_clock<= clock_word(15);

process(slow_clock)

   variable display_selection: std_logic_vector(1 downto 0);
   begin
   if slow_clock'event and slow_clock='1' then

   case display_selection is

      when "00" => temperary_data <=display_A;
      ---- activate LED1 and Deactivate LED2, LED3, LED4. Active low.

         select_Display_A <='0';
         select_Display_B <='1';
         select_Display_C <='1';
         select_Display_D <='1';

         display_selection := display_selection+'1'; --para movernos al siguiente caso

      when "01" => temperary_data <=display_B;
      -- activate LED2 and Deactivate LED1, LED3, LED4. Active low.


         select_Display_A <='1';
         select_Display_B <='0';
         select_Display_C <='1';
         select_Display_D <='1';

         display_selection := display_selection+'1';  --para movernos al siguiente caso

      when "10" => temperary_data <=display_C;
      -- activate LED3 and Deactivate LED1, LED2, LED4. Active low.

         select_Display_A <='1';
         select_Display_B <='1';
         select_Display_C <='0';
         select_Display_D <='1';

         display_selection := display_selection+'1';--para movernos al siguiente caso

      when others => temperary_data <=display_D;
      -- activate LED4 and Deactivate LED1, LED, LED3. Active low.

         select_Display_A <='1';
         select_Display_B <='1';
         select_Display_C <='1';
         select_Display_D <='0';

         display_selection := display_selection+'1';--para movernos al siguiente caso

      end case;
   end if;
end process;
```

4-15 Código principal Módulo 7 Segmentos

5 Integración, Pruebas y Resultados

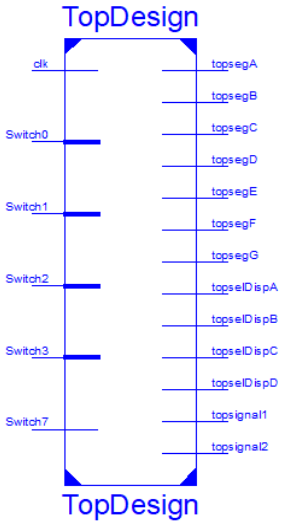
5.1 Integración

5.1.1 Fichero de restricciones UCF

En este fichero de texto se define qué pines físicos de la FPGA se conectan a los puertos de entrada/salida de nuestro circuito. Los nombres utilizados en el fichero UCF para las señales son *clk, 7 segment display y switches.*

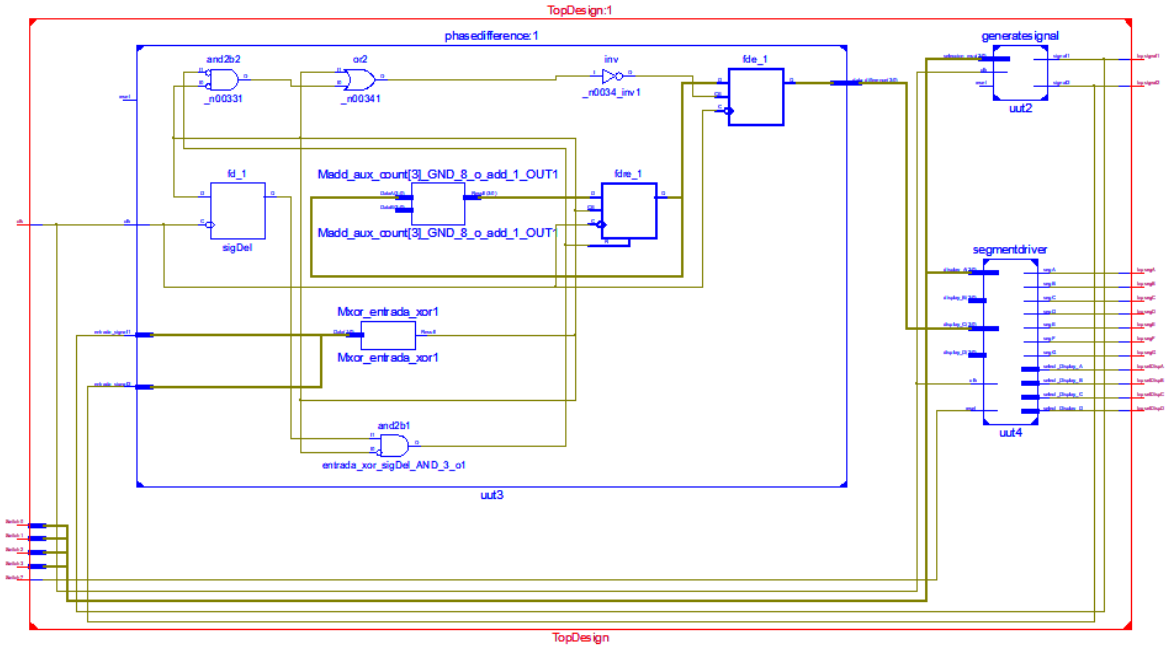
5.1.2 Síntesis

La traducción del código del circuito a puertas lógicas se realiza durante la síntesis, y se obtiene a partir de la herramienta *XST: Xilinx Synthesis Technology*. Es en el sintetizador nativo de Xilinx y se materializa en el ejecutable xst.exe. Genera (entre otros) un fichero de extensión *.ngc*. También se podrá visualizar un esquemático RTL donde puede comprobarse cómo se ha implementado la lógica combinacional.



5-1 Esquema RTL Top Design

Se muestra con más detalle el módulo medidor de desfase:



5-2 Esquema RTL del proyecto

5.1.3 Implementación del Diseño

En este punto se busca generar una fase de traducción dentro de la implementación. Genera el fichero *.ngd*

- **Translate:** Convierte la netlist en un formato binario específico de herramienta.
- **Map:** Asigna los elementos lógicos de lo desarrollado previamente en los elementos específicos como CLBs e IOBs, que realmente implementan funciones lógicas en un dispositivo.
- **Place & Route:** Se asignan bloques de diseño creados durante el mapeo a ubicaciones específicas en la FPGA y se asignan las rutas de interconexión también de la FPGA.

5.1.4 Generate Programming File

El último paso de implementación es generar el archivo de programación para la FPGA. En el proceso, se prepara la información necesaria para la configuración del proyecto, generando los datos en los formatos de archivo necesarios para su familia de dispositivos de destino, como el fichero *.bit* que será el que utilicemos para cargar nuestro trabajo en la FPGA.

5.2 Pruebas

5.2.1 Generación Banco de Pruebas

El TestBench es un set de pruebas que se utiliza para verificar el correcto funcionamiento del proyecto. Así que, se ha intentado cubrir todos los casos posibles y comprobar que las salidas obtenidas son iguales a las esperadas. Para ello se ha añadido un banco de pruebas.

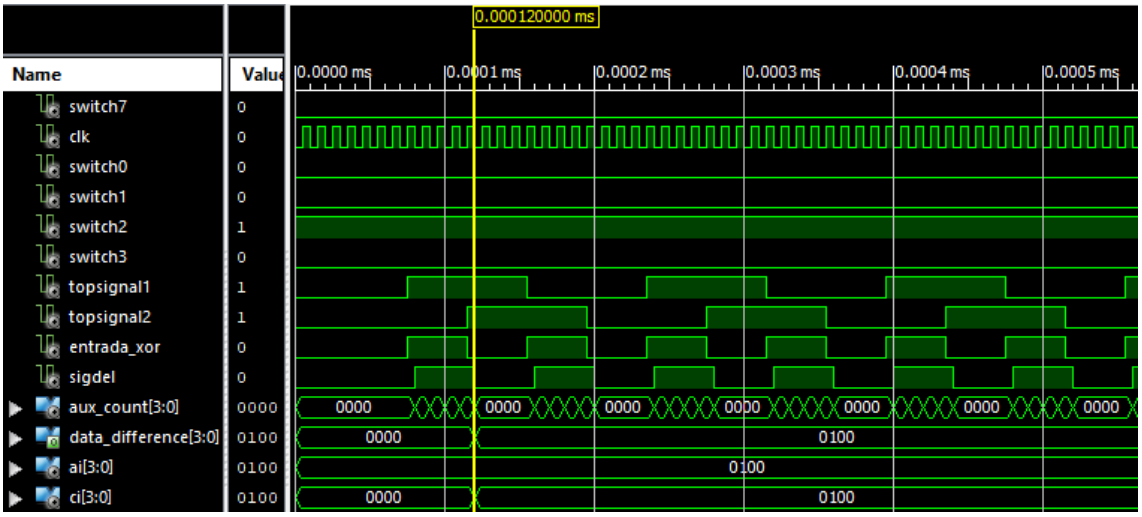
La arquitectura del banco de pruebas consiste en añadir como componente el módulo *TopDesign,* declarar señales internas del banco de prueba (en este caso con el mismo nombre) e ir asignando valores a las señales de entrada para posteriormente realizar la instanciación con un mapa de puertos.

Además de eso se tiene que crea una constante *clk* e igualarla al valor del periodo de la FPGA (10ns) ya que así los diagramas del tiempo de la simulación coincidirán perfectamente con la realidad de la FPGA.

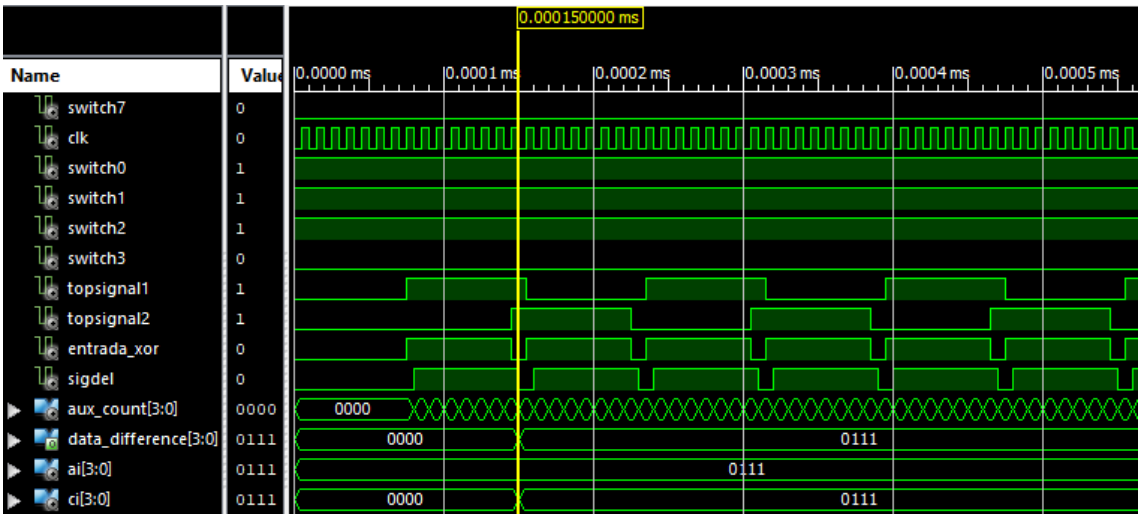
Finalmente se crea el proceso encargado de ejecutar el reloj de forma periódica.

5.2.2 Simulación

A continuación, se muestra el ejemplo de la simulación del proyecto. Ambas figuras siguientes son ejemplos de simulación. En cada una de las imágenes se han utilizado señales de entrada con distinta diferencia de fase.



5-3 Simulación con entrada "0100"



5-4 Simulación con entrada "0111"

En la simulación se aprecia como el *switch7* es el que controla el reset, por lo que estará a cero en todo momento para que se pueda ejecutar la simulación. Posteriormente puede verse que los interruptores marcan el número binario *"100"* para la *figura 5-3* y en la *figura 5-4 "111"*. También se observan las dos señales genedaras con su respectivo desfase y su correspondiente señal XOR que en mi caso he llamado *entrada_xor*.

Por supuesto se muestra la señal *aux_count*, encargada de contar en todo momento los pulsos de reloj en los que la señal XOR es igual a 1. Para actualizar únicamente el valor de la última cuenta se observa lo explicado en el capítulo anterior, específicamente en el módulo medidor diferencia de fase y es que sólo se actualizará el valor de la señal *data_difference* gracias a la señal *sigDel* que es la misma señal *entrada_xor*, pero con una diferencia de medio ciclo de reloj. Es decir, la señal que controla la diferencia de fase se actualizará en el momento en el que *sigDel* cambie de valor de '1' a '0'.

Finalmente, también se muestra en la simulación como *ai* y *ci*, que son los valores que controlan y muestran el valor por los displays, son los mismos y coinciden con el valor de desfase marcado por los interruptores.

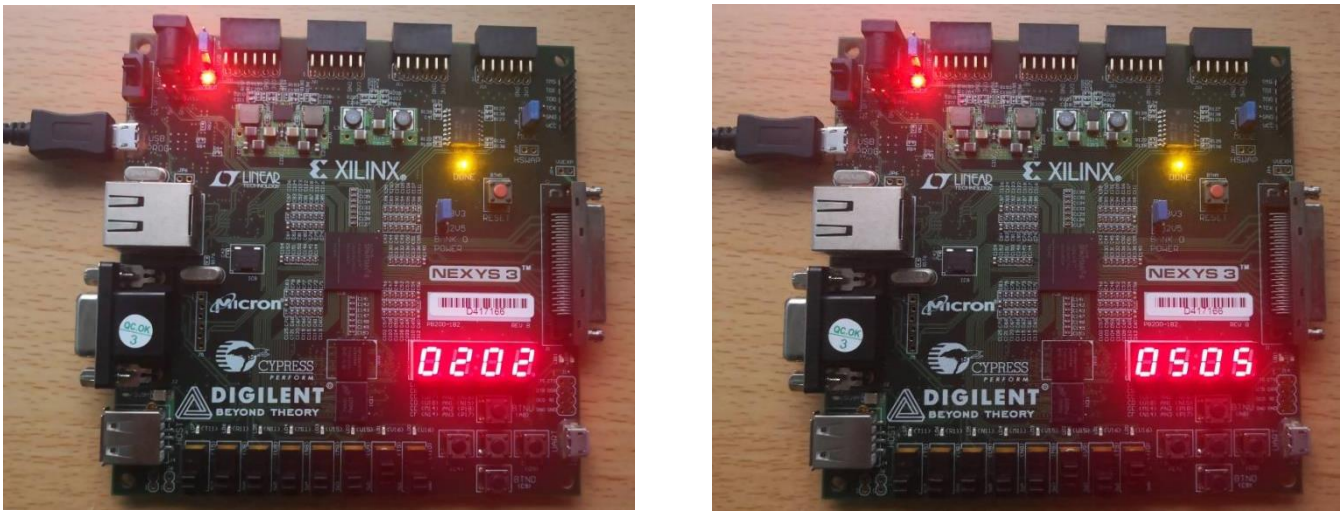
5.2.3 Resultados

Para finalizar, se muestran los resultados obtenidos tras verificar el funcionamiento del sistema sobre la FPGA.

El programa *Digilent Adept* ha sido utilizado para la comunicación con el dispositivo.

En el programa añadiremos el fichero *.bit* generado anteriormente para que después de añadirlo poder programarlo en la placa.

Una vez programado el proyecto en la FPGA, se puede ir eligiendo mediante los interruptores el desfase deseado, mostrándose en la placa de la siguiente manera:



5-5 Resultados para diferentes desfases

Tal y como se esperaba los *displays A* y *C* tienen el mismo valor debido a que el *display A* muestra el desfase elegido y el *display C* muestra el calculado y ambos deben de ser iguales.

Se resalta el hecho de que en este caso, las señales tienen 160ns de período, por lo que, en el caso de superar 8 periodos de diferencia de fase, el resultado de la diferencia de fase se corresponderá con la diferencia de fase absoluta.

6 Conclusiones y Trabajo Futuro

6.1 Conclusiones

Una vez finalizado este proyecto, se ha diseñado un medidor de fase junto con un módulo de pruebas para lo cual se han seguido una serie de pautas o pasos para realizar el proyecto.

El primero ha sido el de conocer en profundidad las herramientas y programas con los que se iban a trabajar, así como conocer la programación en lenguaje VHDL.

El segundo paso dado, fue el de realizar una descripción del diseño de los circuitos, aplicando un diseño jerárquico y dividiendo el trabajo en diferentes módulos en los que cada uno posee una funcionalidad diferente (y a continuación se programó en VHDL los circuitos en diferentes módulos).

El tercer paso fue el de implementación y posterior simulación viendo que los resultados nos coincidían perfectamente con lo que se buscaba, esto es así ya que el módulo de desfase introduce desfases de un mínimo de 10ns, y el sistema de medida de fase tiene precisamente una precisión de 10ns.

Por último se programó la FPGA con el fichero *".bit"* y se comprobó que el diseño funcionaba de forma correcta sobre la FPGA.

El principal contratiempo durante la realización de este proyecto ocurrió al comprobar que cuando no hay desfase, la salida de XOR es constante e igual a cero, por lo que el medidor no realizaba ninguna nueva medición y dejaba almacenado el último valor medido. Para solucionarlo se ha optado por utilizar un *switch* como *reset* manual, para resetear el sistema cada vez que quiera realizarse una nueva medición.

6.2 Trabajo futuro

El trabajo futuro más inmediato sería completar el diseño a partir de la recepción de señales externas a la FPGA.

Otra vía de trabajo sería el de realizar un sistema encargado de medir a partir de un tiempo de integración variable. Por poner un ejemplo, se trataría de un sistema que durante un segundo tome las medidas de las diferencias de fase y luego realice una media entre todos los datos obtenidos para obtener un desfase lo más cercano posible al original ya que en estos casos se podrían trabajar con señales que poseen cierta distorsión o ruido, e incluso posibles imperfecciones en el sistema. Con respecto al tema de precisión cabe destacar que la precisión del medidor de fase del proyecto es de 10ns (igual al periodo de reloj de la FPGA) por lo que podría utilizarse un reloj interno más rápido para aumentar la precisión del sistema.

Referencias

[1] Jean-Pierre Deschamps, Gustavo D.Sutter and Enrique Cantó “Guide to FPGA Implementation of Arithmetic Functions”, 2012.
[2] Pong P.Chu, “FPGA prototyping by VHDL examples”, 2008.
[3] John F.Wakerly “Diseño Digital: Principios y Prácticas”, 2001.
[4] Ricardo Cayssials “Sistemas Embebidos FPGA”, Primera Edición.
[5] Noberto Malpica González de la Vega, Susana Borromeo y Felipe Machado “Diseño Digital con Esquemáticos y FPGA”, Primera Edición.

[6] Sinusoide. Retrieved from *https://es.wikipedia.org/wiki/Sinusoide*

[7] Digital Phase Detector. Retrieved from *https://www.electronics-notes.com/articles/radio/pll-phase-locked-loop/phase-detector-digital-analogue-mixer.php*

[8] DSP Logger MX300. Retrieved from *http://www.semapi.com.ar/mx300/descripcion.php*

[9] Programable. FPGA. Retrieved from https://sites.google.com/site/logicaprogramable/calculadoras/fpga/digilent---nexys-3-spartan-6-fpga-board

[10] Fpga4student. Seven Segment Display. Retrieved from *http://www.fpga4student.com/2017/09/vhdl-code-for-seven-segment-display.html*

[11] Blogspot. Serial Parallel Shift. Retrieved from http://vhdl4u.blogspot.com.es/2010/02/serial-in-parallel-out-shift.html

[12] Blogspot. Multiplexer. Retrieved from http://vhdl4u.blogspot.com.es/2010/02/vhdl-model-of-818-input-multiplexer.html

[13] Wikipedia. Frecuencímetro. Retrieved from https://es.wikipedia.org/wiki/Frecuenc%C3%ADmetro

[14] Forums. Xilinx. Check signal transition. Retrieved from https://forums.xilinx.com/t5/Virtex-Family-FPGAs/How-to-check-for-two-signal-transitions-in-VHDL/td-p/649235

Glosario

FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
DSP	Digital Signal Processor
PLD	Programmable Logic Devices
ASIC	Application Specific Integrated Circuit
CLB	Configurable Logic Blocks
IOB	Input Output Blocks
FF	Flip Flop
LUT	Look Up Table
USB	Universal Serial Bus
IEEE	Institute of Electrical and Electronics Engineers